ELSEVIER

Available online at www.sciencedirect.com

SCIENCE DIRECT®

The Journal of Logic and Algebraic Programming 65 (2005) 36–49

THE JOURNAL OF LOGIC AND ALGEBRAIC PROGRAMMING

www.elsevier.com/locate/jlap

Computable scalar fields: A basis for PDE software ☆

Magne Haveraaen[a,*], Helmer André Friis[b], Hans Munthe-Kaas[a]

[a] *Department of Informatics, University of Bergen, P.O. Box 7800, N-5020 BERGEN, Norway*
[b] *RF—Rogaland Research, P.O. Box 8046, N-4068 Stavanger, Norway*

Received 15 December 2004; accepted 21 December 2004

Abstract

Partial differential equations (PDEs) are fundamental in the formulation of mathematical models of the physical world. Computer simulation of PDEs is an efficient and important tool in science and engineering. Implicit in this is the question of the computability of PDEs. In this context we present the notions of scalar and tensor fields, and discuss why these abstractions are useful for the practical formulation of solvers for PDEs.

Given computable scalar fields, the operations on tensor fields will also be computable. As a consequence we get computable solvers for PDEs. The traditional numerical methods for achieving computability by various approximation techniques (e.g., finite difference, finite element or finite volume methods), all have artifacts in the form of numerical inaccuracies and various forms of noise in the solutions. We hope these observations will inspire the development of a theory for computable scalar fields, which either lets us understand why these artefacts are inherent, or provides us with better tools for constructing these basic building blocks.

1. Introduction

A large portion of present super-computer resources is devoted to the solution of partial differential equations (PDEs). PDEs are used to describe problems ranging from physical phenomena such as electromagnetism and sound waves, via engineering applications such as bridge constructions and air flow around aeroplanes, to economics predictions such as expected return on investment for bonds and shares. The efficient solution of PDE problems is therefore important.

☆ This investigation has been carried out with the support of the European Union, the Research council of Norway (NFR), and by a grant of computing resources from NFR's Supercomputer Committee.

* Corresponding author.
URL: http://www.ii.uib.no/~magne/ (M. Haveraaen), http://www.ii.uib.no/~hans/ (H. Munthe-Kaas).

1567-8326/$ - see front matter © 2005 Elsevier Inc. All rights reserved.
doi:10.1016/j.jlap.2004.12.001

PDEs describe continuous phenomena. Solving PDEs on digital computers, however, requires discretisation of the computational domains involved. Traditionally this discretisation has started already in the formulation of the problem to be solved. A PDE solver then embodies a plethora of concerns: coordinate systems, time integration methods, spatial grids, differentiation stencils, PDE simplification assumptions, convergence acceleration techniques, storage structures, etc. The underlying fundamental concepts are well hidden, and the solver is seemingly concerned with the computation of simple functions on the reals.

Our investigation [1,4,6,7,9] of these issues from a software engineering viewpoint has shown that the concepts of coordinate free mathematics, see, e.g., [15], are more appropriate when it comes to structuring and understanding PDE software. This realization can be carried over to practice with software concepts like abstract data types and object orientation, allowing the separation of "*what*", the external *presentation*, from "*how*", the internal *representation*. For PDEs this means that we may combine abstractions from *pure mathematics* with implementations from *applied mathematics* [13]. The Sophus software architecture [7] which is built on these principles has layers corresponding to

- discrete structures in the form of sequential and parallel meshes,
- continuous structures in the form of scalar fields on manifolds,
- coordinate free structures in the form of tensor fields, and
- PDE solvers with time integrators on coordinate free structures.

This gives a framework for *coordinate free numerics*. Here the transition between the continuous and the discrete is confined to one layer, the scalar fields, given the assumption that the reals may be computed with sufficient precision. Computability of the entire PDE solver can therefore be reduced to computability of the scalar field.

This paper is organised as follows. In section two we discuss abstractions involved in the formulation of coordinate free PDE solvers. Then we present some notions of computable scalar fields and the standard discretisation techniques with a focus on the finite difference method, and show some of the problems that arise when approximating scalar fields. Finally, we conclude with the need for computable scalar fields, hoping to have inspired the development of this notion.

2. Fundamental concepts of PDEs

The concepts involved in the formulation of coordinate free PDEs and PDE solvers are time varying tensor fields. To understand what these are, we will start with the basic building block of a scalar field, then define tensors and tensor fields, and finally introduce time variation. The starting point is the notion of a commutative ring with some additional operations.

2.1. Commutative ring

A *commutative ring R with unit* has operations $+, -, * : R \times R \to R$ and constants $0, 1 :\to R$. The operations are such that $*$ distributes over $+$, $+$ is associative and commutative with 0 as neutral element and $-$ as inverse operation, $*$ is associative and commutative with 1 as neutral element.

In addition we will assume that the ring has a partial operation $/ : R \times R \xrightarrow{p} R$, which, wherever it is defined, is inverse to $*$. Examples of such rings are the real numbers $\mathbb{R}$ and

the complex numbers $\mathbb{C}$. In both these cases / is undefined only if the second argument is 0.

A ring may also have one or more derivation operations $D : R \to R$. A derivation operation obeys the Leibnitz rules,

$$D(a + b) = D(a) + D(b),$$
$$D(a * b) = D(a) * b + a * D(b)$$

for all $a, b \in R$. The derivation operations on $\mathbb{R}$ and $\mathbb{C}$ are trivial, i.e., $D(a) = 0$ for all elements a.

2.2. Scalar fields

The set of *scalar fields*, $\mathrm{SF} = R^X$, can be seen as the functions from some manifold X, an open set with a topology that defines proximity and direction, to a ring R. We will write $a[x]$ for the value of a scalar field $a \in \mathrm{SF}$ at a point $x \in X$. An example of a manifold is an open unit cube in $\mathbb{R}^n$, $n \in \mathbb{N} = \{0, 1, 2, \ldots\}$, i.e., $X = (0, 1)^n$, and we say that X has *dimension* n. A scalar field has point-wise lifting of the operations on the ring R: a ring function $f_R : R \times \cdots \times R \to R$ with k arguments lifts point-wise to a function $f_{\mathrm{SF}} : \mathrm{SF} \times \cdots \times \mathrm{SF} \to \mathrm{SF}$ defined by

$$f_{\mathrm{SF}}(a_1, \ldots, a_k)[x] = f_R(a_1[x], \ldots, a_k[x]) \tag{1}$$

for all points $x \in X$ and scalar fields $a_1, \ldots, a_k \in \mathrm{SF}$. Equational properties on f_R are preserved by the lifting. Specifically the ring operations lift from R to the scalar field, giving the scalar field the properties of a commutative ring with unit and a partial division operation $/ : \mathrm{SF} \times \mathrm{SF} \xrightarrow{p} \mathrm{SF}$ which, wherever it is defined, is inverse to scalar field multiplication. The scalar field division a/b is undefined wherever there exists an $x \in X$ such that the underlying ring division $a[x]/b[x]$ is undefined, thus scalar field division is undefined wherever the second scalar field takes the value 0 in at least one point (assuming the underlying ring is $\mathbb{R}$ or $\mathbb{C}$).

In addition, scalar fields have non-trivial directional derivatives and volume and surface integration operations (for sub-domains of the manifold). For the simple case of the manifold being an open, n-dimensional unit cube, partial derivatives

$$\partial/\partial x^i : \mathrm{SF} \to \mathrm{SF}$$

may be associated with each of the spatial directions $i \in \{1, 2, \ldots, n\}$. Other directional derivatives may be formed by linear combinations (on the ring of scalar fields) of these.

The volume and surface integrals

$$\int_V \_ \, \mathrm{d}(n) : \mathrm{SF} \to R,$$
$$\int_S \_ \, \mathrm{d}(n-1) : \mathrm{SF} \to R$$

(respectively, using _ as a placeholder for the argument), are defined for $V \subseteq X$ and $S \subseteq X$ such that S spans an $(n-1)$-dimensional surface in X. They represent the accumulated value of the argument scalar field in the volume V and surface S, respectively.

2.3. Tensors

A simplified view of *tensors* is as the collection of rank k, $k \in \mathbb{N}$, functions $T^k = S^{(I^k)}$ from some discrete index domain I^k, for $I = \{1, 2, \ldots, m\}$ and $m \in \mathbb{N}$, to a commutative ring S with unit. Operations on tensors are indexed by the ranks of the arguments. The most important operations are the tensor products, tensor additions, and tensor applications.

$$\begin{aligned}
&\otimes_{k,\ell} : T^k \times T^\ell \to T^{k+\ell},\\
&+_k : T^k \times T^k \to T^k,\\
&@_{k,\ell} : T^{k+\ell} \times T^\ell \to T^k.
\end{aligned}$$

Tensors are multi-linear maps. Thus given a rank $k + \ell$ tensor t and a rank ℓ tensor $t' = t_1 \otimes_{0,\ell} (t_2 +_\ell t_3)$, then

$$\begin{aligned}
t\ @_{k,\ell}\ t' &= t\ @_{k,\ell}\ (t_1 \otimes_{0,\ell} (t_2 +_\ell t_3))\\
&= t_1 \otimes_{0,k} ((t\ @_{k,\ell}\ t_2) +_k (t\ @_{k,\ell}\ t_3)).
\end{aligned}$$

Tensors with $k = 0$ are called scalar tensors, and are in one-to-one correspondence with the underlying ring S by the isomorphisms

$$\begin{aligned}
&\mathrm{inj} : S \to T^0,\\
&\mathrm{pro} : T^0 \to S.
\end{aligned}$$

When $k = 1$ the tensors are called vectors (of dimension m), and a set of m linearly independent vectors are called basis vectors. Given a basis, then all tensor values can be created from linear combinations (using $\otimes$ and $+$) of scalar tensors and the basis vectors.

For scalar tensors operations $\otimes_{0,0}$ and $@_{0,0}$ coincide, and we also have an inverse $/ : T^0, T^0 \xrightarrow{p} T^0$ to $\otimes_{0,0}$ given by $\mathrm{pro}(t/t') = \mathrm{pro}(t)/\mathrm{pro}(t')$.

2.4. Tensor fields

If the underlying ring S for the tensors T^k is the scalar fields $S = \mathrm{SF} = R^X$ with dimensions $n = m$ for the manifold X, we get a *tensor field*. Tensor fields are tensors with the tensor operations. They also have a plethora of derivation and integration operations. Derivation operations include gradient, divergence and Lie derivatives

$$\begin{aligned}
&\nabla : T^k \to T^{k+1},\\
&\nabla\cdot : T^{k+1} \to T^k,\\
&\mathscr{L} : T^1 \times T^k \to T^k.
\end{aligned}$$

Only scalar tensor fields have integration operations, for volumes and surfaces,

$$\begin{aligned}
&\int_V _\ \mathrm{d}(n) : T^0 \to R,\\
&\int_S _\ \mathrm{d}(n-1) : T^0 \to R,
\end{aligned}$$

where $V \subseteq X$ is a volume and $S \subseteq X$ is a surface in X. Neither the derivation nor the integration operations are liftings of the corresponding scalar field functions. These tensor field operations are constructed from the scalar field operations, and take into account the geometry of the problem domain, such as the number of dimensions, rotational or translational symmetry, or curvilinearity. This awareness of the coordinate system makes the use of tensor fields completely coordinate free.

2.5. Time varying scalar and tensor fields and PDEs

Time varying behaviour can be described by considering *time fields* $\mathrm{TF} = \mathrm{SF}^Y$, scalar fields SF in time Y, where $Y \subseteq \mathbb{R}^1$ is an open interval, e.g., $Y = (0, 1)$, and $\mathrm{SF} = R^X$ are the scalar fields in space X. The operations on the time fields TF come from three sources.

- The ring operations which are lifted from the ring R via SF.
- The spatial derivatives and integrations which are lifted from SF.
- The temporal scalar field operations, notably the partial derivative $\partial/\partial t : \mathrm{TF} \to \mathrm{TF}$ in time and the volume integral $\int_{\Delta Y} \_\, \mathrm{d}(1) : \mathrm{TF} \to \mathrm{SF}$ for time intervals $\Delta Y \subseteq Y$.

We may now do a tensor field construction $U^k = \mathrm{TF}^{(I^k)}$ on time fields TF with respect to the lifted ring and spatial operations from SF and plainly lift the temporal operations from the time fields TF to the tensors U^k, giving us the apparatus to formulate coordinate free PDEs. One example is the wave equation for elastic materials,

$$\frac{\partial}{\partial t}\left(\frac{\partial}{\partial t}u\right) = \frac{1}{\rho}\nabla\cdot(\Lambda \mathbin{@_{2,2}} \mathscr{L}_u(g)) +_1 f, \tag{2}$$

where $u \in U^1$ is the time varying wave displacement vector, $\rho \in U^0$ is the density of the material, $\Lambda \in U^4$ is the elasticity of the material (Hooke's tensor), $g \in U^2$ describes the geometry of the domain (metric tensor), and $f \in U^1$ is a time varying wave source vector. Normally ρ, Λ and g will remain constant in time (although varying in space). PDEs are often classified as elliptic, parabolic or hyperbolic (or a combination of these) according to their properties. The elastic wave equation (2) is hyperbolic.

As can be seen, the elastic wave equation relates the change in the wave displacement (in time) with its spatial distribution. Relating change in time with spatial distribution is a characteristic of PDEs.

3. Implementing PDE solvers

PDE solvers are of two main variants: the steady state solvers and the time varying solvers. The steady state solvers try to find what the spatial distribution of the time-varying tensor fields looks like at a time when the rate of change is neglectable. The time varying solvers show how the values of the time-varying tensor fields change in time and space.

In order to implement PDE solvers, we will need implementations of tensor fields (with their operations) and scalar fields (with their operations).

3.1. Tensor field implementations

Tensor fields turn out to be easy to implement. Rank k tensors $T^k = \mathrm{SF}^{(I^k)}$, $I = \{1, 2, \ldots, m\}$, may easily be stored using arrays with k indices, each index ranging over I,

with scalar fields SF as the element type. Tensor operators need to traverse the elements of these arrays computing new tensor values, but the arrays are finite and the expressions computed are finite scalar field expressions. Thus tensor operations are computable if, and only if, the scalar field operations are computable.

3.2. Scalar field implementation requirements

Implementing scalar fields $\mathrm{SF} = R^X$, whether temporal or spatial, turns out to be a much harder problem. They are continuous structures dependent on both the ring R, typically $R = \mathbb{R}$, the real numbers, and the domain X, typically $X = (0, 1)^n \subseteq \mathbb{R}^n$, the unit cube of n dimensions. Using standard ideas of computability on reals [21,22], we require a computable approximation $a : \mathbb{N} \to \mathrm{SF}$ to a scalar field $a \in \mathrm{SF}$ to satisfy

$$\forall p \in \mathbb{N}: \quad \int_X \mathrm{abs}(a(p) - a)\,\mathrm{d}(n) < \frac{1}{2^p}. \tag{3}$$

This has the drawback of having to introduce the absolute value function $\mathrm{abs} : \mathrm{SF} \to \mathrm{SF}$ on scalar fields and also involves taking the integral over the whole scalar field. Observing that we often only need the value of the scalar field at chosen points $x \in X$, we may suggest a point-wise computability requirement,

$$\forall p \in \mathbb{N},\ x \in X: \quad \mathrm{abs}(a(p)[x] - a[x]) < \frac{1}{2^p}. \tag{4}$$

This formulation implies (3), but only involves operations on the underlying ring. This also hints on the possibility of lifting, see Eq. (1), computable ring operations to become computable scalar field operations. The requirement (4) implies that a sum (product, etc.) of two computable scalar fields can be made close enough to the corresponding sum (product, etc.) of the approximated scalar fields, if necessary by iterating the computable argument scalar fields finitely many steps more than the sum (product, etc.) scalar field. For instance, define an approximation to the sum scalar field $(a + b)$ by

$$(a + b)(p) = a(p + 1) + b(p + 1),$$

then

$$\begin{aligned} \forall p \in \mathbb{N},\ x \in X: \quad & \mathrm{abs}((a + b)(p)[x] - (a + b)[x]) \\ & = \mathrm{abs}(a(p + 1)[x] + b(p + 1)[x] - a[x] - b[x]) \\ & < \frac{1}{2^{p+1}} + \frac{1}{2^{p+1}} = \frac{1}{2^p} \end{aligned}$$

showing the (pointwise) computability of (the lifted) $+$.

We must also consider the non-lifted scalar field operations, i.e., the derivation and integration operations. For the volume integration operation $\int_V \_\,\mathrm{d}(n)$, the requirement (3) implies that the integral of a computable scalar field can be made close enough to the integral of the approximated scalar field for any volume $V \subseteq X$. The stronger requirement (4) ensures this for the surface integral $\int_S \_\,\mathrm{d}(n)$ for any surface $S \subseteq X$.

But for the derivation operation this may not be the case. Imagine a flat scalar field, like the lifted 0 scalar field. Any derivative of this scalar field will be 0 at any point $x \in X$. If the computable approximation $a(p)$ is a fluctuating scalar field, the derivative may become arbitrarily large at some points, even though the computable scalar field itself never strays more than $\frac{1}{2^p}$ from its correct value. Thus we need to add the additional requirement that, for every derivation operation D,

$$\forall p \in \mathbb{N}: \quad \int_X \mathrm{abs}(D(a)(p) - D(a))\mathrm{d}(n) < \frac{1}{2^p}, \tag{5}$$

or in the pointwise formulation,

$$\forall p \in \mathbb{N}, x \in X: \quad \mathrm{abs}(D(a)(p)[x] - D(a)[x]) < \frac{1}{2^p}. \tag{6}$$

Since X is a continuous domain, we also need to take into account the computable approximation $x : \mathbb{N} \to X$ of a chosen point $x \in X$, i.e., $\forall q \in \mathbb{N}$: $\mathrm{abs}(x(q) - x) < 1/2^q$. Then we may formulate computability requirements for a scalar field a at any point x,

$$\forall p \in \mathbb{N}, \ \exists Q \in \mathbb{N}, \ \forall q \in \mathbb{N}, \ q > Q: \quad \mathrm{abs}(a(p)[x(q)] - a[x]) < \frac{1}{2^p}, \tag{7}$$

$$\forall p \in \mathbb{N}, \ \exists Q \in \mathbb{N}, \ \forall q \in \mathbb{N}, \ q > Q: \quad \mathrm{abs}(D(a)(p)[x(q)] - D(a)[x]) < \frac{1}{2^p}. \tag{8}$$

Note that the approximation of $x(q)$ to x may be quite coarse yet we still achieve a good approximation of $a[x]$ and $D(a)[x]$, e.g., if the scalar field a is fairly flat in the neighbourhood of x.

3.2.1. Temporal scalar fields

The requirements on the temporal scalar field are different from those on the spatial scalar field. Normally a PDE like (2) describes causal relationships, so a simulation will start at a given point in time with a known spatial distribution, and compute the unknown u forward in time. We need to do integration forward in time based on our knowledge of u backward in time. This is often handled by doing a few steps of the Taylor series expansion of the PDE—discretisation in time—, and then rephrasing the PDE so that the value of a future time-step depends on known information about previous time-steps. The time-steps are then given by a computable number $\Delta T \in \mathbb{R}$ which represents the interval for which time integration takes place. Time steps can often be chosen such that they coincide (to a sufficient degree) with the chosen temporal $t \in (0, 1) \subseteq \mathbb{R}^1$ coordinates of the space-time points where we want answers. More sophisticated time integration techniques exist, see [9] for some developments.

3.2.2. Spatial scalar fields

Implementing spatial scalar fields is a well-developed area of numerics, and the main approaches are finite difference methods (FDM), finite element methods (FEM) and finite volume methods (FVM). FEM and FVM are based on a subpartitioning of the spatial domain X, and the scalar field is approximated by functions of a known form (called *form functions*), typically computable real polynomials of a fixed degree, within each partition. This has the drawback that many operations on the scalar fields, such as multiplication and derivation, invalidate the representation: the product of two polynomials has a degree equal to the sum of the argument polynomial degrees, and repeated differentiation of a polynomial will yield the constant 0 after a few iterations. To circumvent this, PDE solvers use spatial integrals at each time step to compound as much information as possible about each partition, and then recreate a scalar field of the appropriate representation before the next iteration in time.

The FDM is simpler, but is best suited to solve problems on simply shaped domains X, such as a rectangular domain. Typically a scalar field $a \in \mathrm{SF}$ is sampled at regular

intervals in each direction in $X = (0, 1)^n$, yielding a multi-dimensional array with n indices of computable real values. The values $a[x]$ and $D(a)[x]$ at a point $x \in X$ is determined by interpolation functions. The choice of interpolation function is based on knowledge of the form of the scalar field. In most cases a simple linear interpolation suffices for the value $a[x]$, while more elaborate schemes taking into account several neighbouring sampling points are used for the spatial derivatives. In SeisMod [7] we use linear interpolation between the nearest sampling points for the point values $a[x]$ and use 9 sampling points in each spatial direction for the partial derivatives $D(a)[x]$. For a given interpolation, the accuracy depends on the density of the sampling points. The ring operations on the scalar field are lifted from the underlying ring as point-wise operations on the array representation. The partial derivatives are implemented using weighted sums of differences between neighbouring points in the array representation, and integrations are done by weighted sums of the sampled values. Thus the sampled values may also be thought of as a step function approximation to the integrals [26]. The FDM approximation is known as a good spatial approximation for the elastic wave PDE.

3.3. Numerical accuracy considerations

One important observation, presented, e.g., in [17], is that the coarseness of the sampling for the FDM, or of the subpartitioning of the spatial domain for the FEM and the FVM, far out-ways inaccuracies in any reasonable approximation to the computable reals. Thus normal floating point numbers [5] in most cases are more than precise enough compared to the spatial discretisation. Increasing the accuracy of, e.g., an FDM approximation, basically requires increasing the number of sampling points. At some stage an increase in the accuracy of the real approximation is needed.

Unfortunately, there are many phenomena related to a numerical discretisation which will have a larger impact on the accuracy of a PDE solver than discretisation resolution alone. The rest of this section discusses some of these issues and the compromises a numerical analyst makes between these. We use the SeisMod FDM solver [7,3] for the elastic wave equation (2) to illustrate the concepts. The SeisMod solver starts by inputting time parameters, the density ρ and elasticity Λ tensors for a given spatial resolution ΔX, then it is given a characterisation of the time varying wave source vector f. This is typically a point source, i.e., a Dirac delta function characterised by a spatial coordinate and frequency, and has an amplitude which rapidly decreases to 0 in time and space. The simulation starts with the spatial tensor field $u = 0$, and continues for the given number of time steps at a given time resolution ΔT.

3.4. Run-time constraints

A run-time constraint is not connected to the accuracy of the numerical simulation of a PDE solver per se, but goes to the heart of the usefulness of such solvers. The first observation is that if we halve the time resolution (by halving ΔT), then the run-time of the solver doubles. Likewise, if we halve the spatial resolution (by halving ΔX), run-time increases by a factor 8 for a three-dimensional problem. Luckily, we may in some cases simplify the *geometry* of the problem and reduce a three-dimensional problem to one needing only two-dimensional scalar field data sets. This happens if we have translation symmetry (when the whole problem may be treated as two-dimensional) or rotational symmetry (when only

the spatial scalar fields may be simplified to two dimensions). With these symmetries the partial derivative in one of the directions becomes trivial, i.e., 0 everywhere. We may then eliminate computing the trivial derivative and all expressions where the trivial derivative is a factor.

The savings we get on computational resources by exploiting such properties are significant. In the elastic wave equation (2) the rank 4 tensor field Λ in principle has 81 components in three-dimensions. But properties of the geometry of three-dimensional space reduces this to 21 distinct components and implies other symmetry properties for the other tensors, in effect more than halving the computational cost of solving the equation. If we are able to exploit translation symmetry, effectively reducing the problem to two-dimensions, we save another 30% on computing the derivatives $\nabla\cdot$ and $\mathscr{L}$ alone. Exploiting more specific symmetry properties of the data sets may further significantly reduce the cost of storing data and doing computations, see [3] for details. However, the really big savings do not come from the tensor level properties. Consider the case of a three-dimensional scalar field with 1000 sampling points in each direction, i.e., 10^9 sampling points all together. If we manage to reduce this to a two-dimensional data set, we will only need 10^6 sampling points, reducing our data requirements and computation times by a factor of 1000.

For a single simulation, a long run-time of several days may be tolerable. Often the solution of a PDE like (2) is part of an *inverse problem*. In an inverse problem the value of the tensor u in Eq. (2) may be known at selected space-time coordinates, and the purpose of solving the PDE is to find the values of ρ and Λ. It is impossible to use the PDE directly for this purpose, so instead values of ρ and Λ are estimated. The PDE is simulated for the chosen time fragment, and the simulated u values are compared with the known u value fragments. Then the estimates of ρ and Λ are adjusted based on the found mismatches with the known parts of u, and a new simulation is performed. The whole process is repeated until a good enough estimate is achieved. Overnight run-times of 12–16 h are acceptable for such problems, as this gives a full working day for re-estimating the parameters ρ and Λ between simulations. A doubling of run-time will easily move simulation time from acceptable to unacceptably long. People therefore tend to use a resolution which is as coarse as possible for the needed accuracy of the result.

3.5. Stability

So far we have presented the spatial and the temporal resolutions as independent. We have also treated spatial resolution as independent of the data and problem we are working on. In practice all of these must be considered together [18]. The spatial resolution depends very much on the variability of the scalar field. If the scalar fields are almost constant, a coarse spatial resolution may be used. If there is great variation over a short distance, the spatial resolution must be small enough to capture these details. So ΔX is often more sensitive to the variability of the input data than to the desired accuracy. Further, for hyperbolic PDEs like (2), maximal time resolution is dependent on the spatial resolution, i.e., $\Delta T \leqslant c\Delta X$ for some positive number $c \in \mathbb{R}$. If ΔT is larger than this threshold, the computations become unstable, and very soon may yield extremely large (increase in energy) or other physically impossible values [18]. A halving of spatial resolution for a three-dimensional problem thus incurs a 16-fold increase in run-time, e.g., from 1 h to 16 h for a simulation.

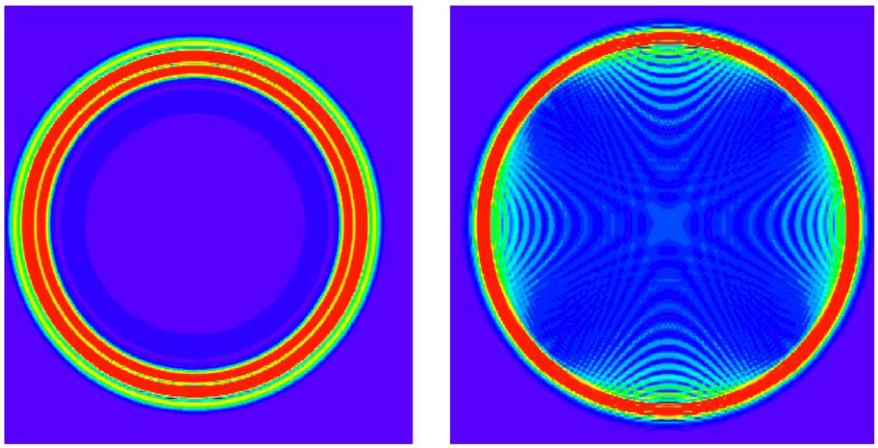

Fig. 1. In the left snapshot the source has frequency 30 Hz, in the right 100 Hz. Numerical dispersion is clearly visible inside the wavefront in the right snapshot. Also notice that the 100 Hz wavefront defines a slightly larger circle than the 30 Hz wavefront. This is also a numerical dispersion phenomenon. Homogeneous medium with 1×1 km grid and 5 m resolution. Snapshots are taken after 175 ms (0.5 ms resolution). Wave source of type Ricker (zero) is placed at the centre.

3.6. Numerical dispersion

Numerical dispersion is an artifact of the FDM [18], where a uniform wavefront disperses into a band of waves. This phenomenon is caused by the discretisation giving different wave frequencies different speeds in the medium, so that some waves move ahead of the wavefront while others lag behind. Some FDM variants which reduce this problem are described in [8,10]. Even if a method like [8] is used to reduce dispersion, having a wave source f with a too high frequency for the spatial resolution will induce this effect, see Fig. 1.

3.7. Modifying spatial resolution

One technique for saving simulation run-time would be to adapt the resolution during the computation in order to meet the minimum requirements of approximation accuracy. The definition of computable scalar fields given in Eq. (4) also implies that the resolution needs to be finer in earlier steps and may be coarser in later steps in order to achieve a given level of accuracy. But modification of spatial resolution during a simulation introduces numerical artifacts. To avoid these, numerical filters must be introduced as part of the resolution modification process. What kind of filters depend on the problem being simulated and what kind of changes in resolution are taking place.

3.8. Boundary handling

Since we only can simulate a very small portion of the physical reality, all PDE solvers are challenged with the problem of how to handle the boundaries. In our example we have two kinds of boundaries: a free surface on the top, and open boundaries on the three remaining sides.

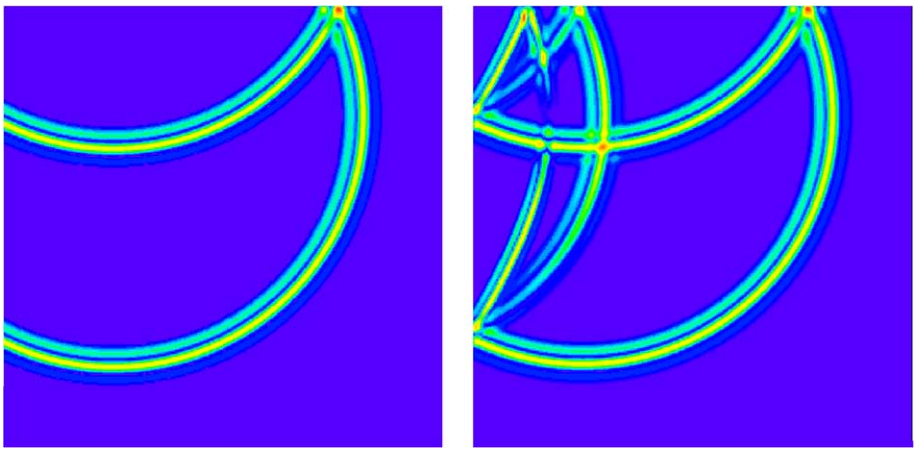

Fig. 2. On the left a large virtual grid without damping ($\gamma = 0$) extends far beyond the visible area, so only the free surface reflection from the top is seen. On the right, a smaller virtual grid with strong damping of the waves ($\gamma = 0.5$) is used. This induces "reflection noise" from the sides into the simulated area. Homogeneous medium with 1×1 km grid, 5 m resolution, and 30 Hz Ricker (zero) wave source in the centre of the upper left quadrant. Snapshots are taken after 250 ms simulation (0.5 ms resolution).

The free surface represents an outer surface of the elastic medium, e.g., where a rock or water medium meets air, which causes a complete reflection of the waves hitting this boundary. Refs. [23,12] present techniques for free surface handling in FDM.

The open boundaries represent a continuation of the material beyond the simulated area, and waves are expected to disappear off into the distance when passing these boundaries. Handling this without resorting to actually extending the simulated area indefinitely is difficult, see [14,19] for an overview. One approach is a so-called local absorbing boundary condition technique, i.e., to give the material beyond the simulated area properties like a ball of cotton [2], absorbing the waves and eliminating any reflection phenomena caused by the numerical simulation. The tuning of this is very difficult, and a too strong damping induces wave reflection, see Fig. 2. There we are using a damping function of the form $k\,\mathrm{e}^{-\gamma^2 x^2}$ where k and γ are constants and x is a measurement of the distance into the damping zone.

3.9. Residual wave sources

In a FDM the border between layers with different properties has to be approximated by the sampling points. For horizontal or vertical borders, this may mean a shifting of the actual border to the nearest sampling points. For slanting or curving borders, the sampling will yield an irregular border which is piecewise horizontal (or vertical) in-between "jumps" on the order of one spatial resolution. The numerical methods tend to leave a small wave residue at such jumps, artificially introducing wave source functions (hot spots). Fig. 3 illustrates this with a close to horizontal border between two layers with different elastic coefficients. Techniques for reducing this problem exist, see [16,11]. Another possibility is using curvilinear grids to straighten out the slanting and curved borders. This may not always be possible in practice, e.g., when two borders meet within the simulated area.

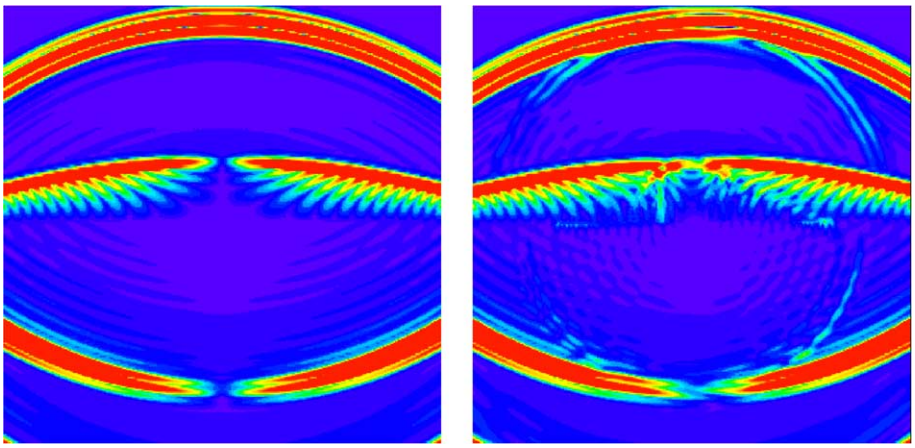

Fig. 3. In the left image, the border between the two layers is horizontal. In the right image the border slants 5 m from lower left to upper right, giving a discontinuous jump of 1 spatial grid-point at the centre of the image. This discontinuity induces a residual wave source every time a wave passes, see the circular wave appearing behind each wavefront in the right image. In addition, numerical dispersion is visible behind the wavefronts in both snapshots. Two-layer medium with 1×1 km grid with 5 m resolution and boundary between the two layers half way down the simulated area. Ricker (zero) 30 Hz source wave at top centre. Snapshot taken after 500 ms simulation (0.5 ms resolution).

3.10. Long-time simulations

If simulated time is very long, the shape of the simulated wave will start to differ noticeably from the actual wave, due to effects of the discretisation itself. Ref. [24] gives a review of methods especially suited for long-time simulations of waves, while [25] develops some efficient techniques to reduce this problem.

4. Conclusion

We have presented abstractions that are central in the formulation of partial differential equations (PDEs) and numerical solvers for PDEs. We have shown that the set of scalar fields $\mathrm{SF} = R^X$ is the natural concept for continuous structures in PDEs, and that a notion of computability for PDE simulations hinges on notions of computable scalar fields. A scalar field is an algebraic structure, and computability of the whole solver then extends from the scalar field along the lines established in [20].

Although we did not suggest any theory of computable scalar fields, we presented some standard approaches to scalar field implementations. This discussion emphasised the observation that the accuracy of a PDE simulation is much more contingent on the resolution of the scalar field discretisation than on the accuracy of the real numbers [17], and that ordinary floating point numbers in many cases are more than accurate enough for simulating a PDE. Further, we pointed out a series of problems with existing discretisation techniques, both as a warning against too simple approaches to computable scalar fields, but also as an inspiration for the definition of an appropriate notion.

The question of what a good definition of a computable scalar field is, and how to implement arbitrary precision scalar fields, still remains open.

Acknowledgment

Thanks to Åsmund Drottning at NORSAR, Thormøhlensgt. 55, N-5008 Bergen, Norway, for providing data models.

References

[1] K. Åhlander, M. Haveraaen, H. Munthe-Kaas, On the role of mathematical abstractions for scientific computing, in: R.F. Boisvert, P.T.P. Tang (Eds.), The Architecture of Scientific Software, Kluwer Academic Publishers, Boston, 2001, pp. 145–158.
[2] C. Cerjan, D. Kosloff, R. Kosloff, M. Reshef, Short note: A nonreflecting boundary condition for discrete acoustic and elastic wave equations, Geophysics 50 (1985) 705–708.
[3] H.A. Friis, T.A. Johansen, M. Haveraaen, H. Munthe-Kaas, Å. Drottning, Use of coordinate free numerics in elastic wave simulation, Appl. Numer. Math. 39 (2) (2001) 151–171.
[4] P.W. Grant, M. Haveraaen, M.F. Webster, Coordinate free programming of computational fluid dynamics problems, Sci. Programming 8 (4) (2000) 211–230.
[5] D. Goldberg, What every computer scientist should know about floating-point arithmetic, ACM Comput. Surveys 23 (1) (1991) 5–48.
[6] M. Haveraaen, Case study on algebraic software methodologies for scientific computing, Sci. Programming 8 (4) (2000) 261–273.
[7] M. Haveraaen, H.A. Friis, T.A. Johansen, Formal software engineering for computational modelling, Nord. J. Comput. 6 (3) (1999) 241–270.
[8] O. Holberg, Computational aspects of the choice of operator and sampling interval for numerical differentiation in large-scale simulation of wave phenomena, Geophys. Prospect. 35 (1987) 629–655.
[9] A. Iserles, H.Z. Munthe-Kaas, S.P. Nørsett, A. Zanna, Lie-group methods, Acta Numer. 9 (2000) 215–365.
[10] S.K. Lele, Compact finite difference schemes with spectral-like resolution, J. Comput. Phys. 103 (1992) 16–42.
[11] F. Muir, J. Dellinger, J. Etgen, D. Nichols, Short note: Modeling elastic fields across irregular boundaries, Geophysics 57 (9) (1992) 1189–1193.
[12] R. Mittet, Free surface boundary conditions for elastic staggered-grid modelling schemes, Geophysics 67 (5) (2002) 1616–1623.
[13] H. Munthe-Kaas, M. Haveraaen, Coordinate free numerics—closing the gap between 'pure' and 'applied' mathematics? Z. Angew. Math. Mech. 76 (suppl. 1) (1996) 487–488.
[14] W.A. Mulder, Experiments with Higdon's absorbing boundary conditions for a number of wave equations, Comput. Geosci. 1 (1) (1997) 85–108.
[15] Bernard Schutz, Geometrical Methods of Mathematical Physics, Cambridge University Press, 1980.
[16] J.S. Sochacki, J.H. George, R.E. Ewing, S.B. Smithson, Interface conditions for acustic and elastic wave-propagation, Geophysics 56 (2) (1991) 168–181.
[17] S. Smale, Some remarks on the foundations of numerical analysis, SIAM Rev. 32 (2) (1990) 211–220.
[18] J.C. Strikwerda, Finite Difference Schemes and Partial Differential Equations, Chapman & Hall, 1989.
[19] S.V. Tsynkov, Numerical solution of problems on unbounded domains. A review, Appl. Numer. Math. 27 (4) (1998) 465–532.
[20] J.V. Tucker, J.I. Zucker, Computable functions and semicomputable sets on many-sorted algebras, in: S. Abramsky, D.M. Gabbay, T.S.E. Maibaum (Eds.), Handbook of Logic in Computer Science, vol. 5, Oxford University Press, Oxford, UK, 2000, pp. 317–523.
[21] K. Weihrauch, Computability, EATCS Monographs on Theoretical Computer Science, vol. 9, Springer-Verlag, Berlin, 1987.
[22] K. Weihrauch, Computable analysis—an introduction, Texts in Theoretical Computer Science—An EATCS Series, Springer-Verlag, Berlin, 2000.
[23] H. Wu, J.M. Lees, Boundary conditions on a finite grid: Applications with pseudospectral wave propagation, Geophysics 62 (5) (1997) 1544–1557.
[24] D.W. Zingg, Comparison of high-accuracy finite-difference methods for linear wave propagation, SIAM J. Sci. Comput. 22 (2) (2000) 476–502.

[25] D.W. Zingg, H. Lomax, H. Jurgens, High-accuracy finite-difference schemes for linear wave propagation, SIAM J. Sci. Comput. 17 (2) (1996) 328–346.
[26] N. Zhong, K. Weihrauch, Computable analysis of partial differential equations, in: CCA'01, Dagstuhl seminar 01461, November 2001.